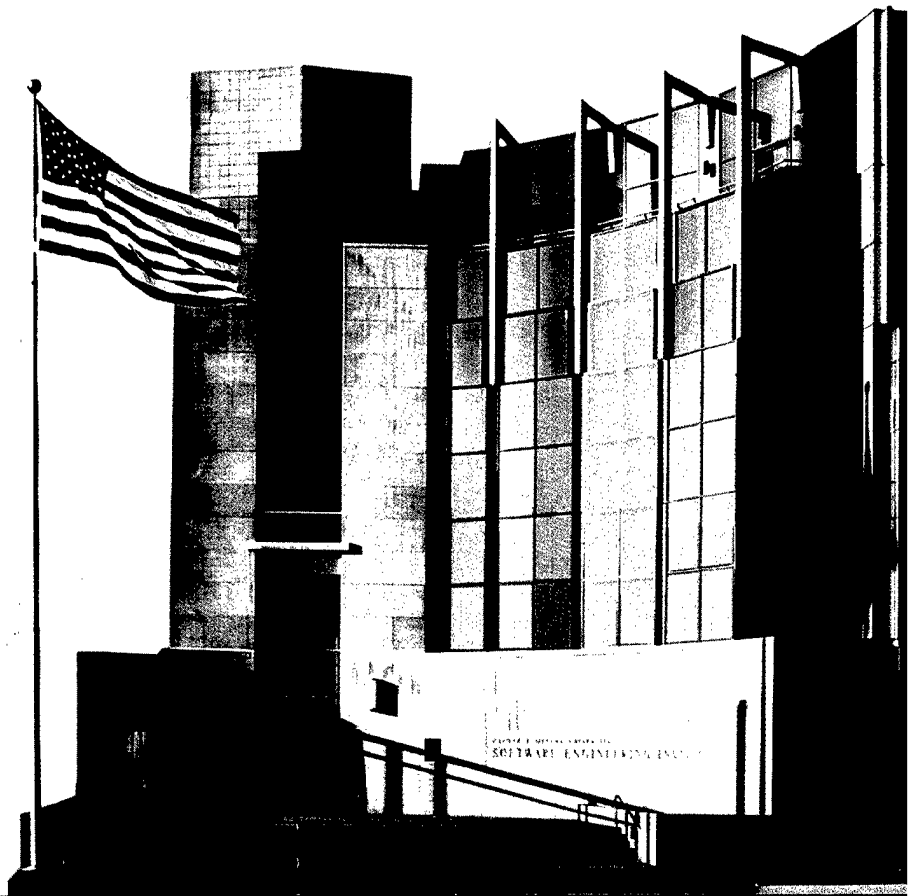
Carnegie Mellon
Software Engineering Institute

Documenting Software Architectures in an Agile World

Paul Clements
James Ivers
Reed Little
Robert Nord
Judith Stafford

July 2003



TECHNICAL NOTE
CMU/SEI-2003-TN-023

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Documenting Software Architectures in an Agile World

CMU/SEI-2003-TN-023

Paul Clements
James Ivers
Reed Little
Robert Nord
Judith Stafford

July 2003

Architecture Tradeoff Analysis Initiative

Table of Contents

List of Figures

Abstract

This report compares the Software Engineering Institute's Views and Beyond approach for documenting software architectures with the documentation philosophy embodied in agile software-development methods. This report proposes an approach for capturing architecture information in a way that is consistent with agile methods.

1 Introduction

This report is the fifth in a series on documenting software architectures.[1] The Software Engineering Institute (SEI[SM])[2] has developed a comprehensive approach to capturing architectural design decisions, loosely called the Views and Beyond (V&B) approach. This technical note will explore the relationship between the V&B approach (which prescribes capturing a rich set of information about the architecture) and so-called "agile" approaches (which emphasize a minimalist, "just in time" approach to documentation).

1. The previous four reports in this series dealt with documenting a layered architecture [Bachmann 00], documenting software behavior [Bachmann 02a] and interfaces [Bachmann 02b], and the overall structure of an architecture documentation package [Bachmann 01]. The previous reports culminated in the publication of a book on software architecture documentation in the Addison-Wesley SEI Series on Software Engineering [Clements 02].

2. SEI is a service mark of Carnegie Mellon University.

2 The V&B Approach

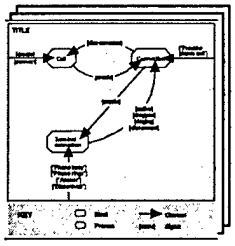
The fundamental principle of the V&B approach (as stated in the book titled *Documenting Software Architectures: Views and Beyond*) is that documenting a software architecture is a matter of documenting the relevant views and then documenting the information that applies across the views [Clements 02]. View-based documentation has emerged as the "best of breed" approach for dealing with software architectures. Some practitioners prescribe a fixed set of views. The Rational Unified Process (RUP), for example, is built on Kruchten's 4+1 approach to creating and capturing software architectures [Kruchten 00]. The Siemens Four Views approach is another example of an approach that suggests a standard view set [Hofmeister 00].

Recently, however, these approaches have been generalized because (as David Parnas pointed out decades ago [Parnas 01]) software systems are characterized by an almost unlimited set of distinct structures that we capture with views. This allows architects the freedom to choose the views that are most relevant to the intended uses of the architecture—for example, performance engineering, change impact analysis, or implementation guidelines. The IEEE recommended best practice for documenting architectures of software-intensive systems (IEEE Std 1471-2000) recognizes this trend by prescribing the conscious selection of views based on stakeholders' concerns [IEEE 00]. The V&B approach also embraces stakeholder-based view selection, but extends the concept to recognize that documentation beyond views is also essential to provide holistic insight into the overall design approach embodied by the architecture.

Figures 1 and 2 give standard outlines for documenting an architectural view and for documenting information beyond views, respectively [Clements 02, Ch. 10].

Views

Section 1. Primary Presentation of the View



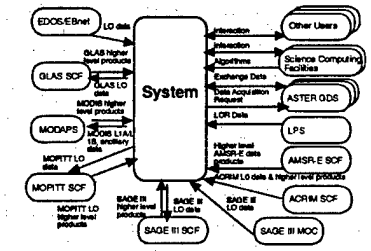
OR

Textual version of the primary presentation

Section 2. Element Catalog

- **Section 2.A Elements and Their Properties**
- **Section 2.B Relations and Their Properties**
- **Section 2.C Element Interfaces**
- **Section 2.D Element Behavior**

Section 3. Context Diagram



Section 4. Variability Guide

Section 5. Architecture Background

- **Section 5.A Design Rationale**
- **Section 5.B Analysis Results**
- **Section 5.C Assumptions**

Section 6. Glossary of Terms

Section 7. Other Information

Figure 1: V&B Outline for Documenting a View

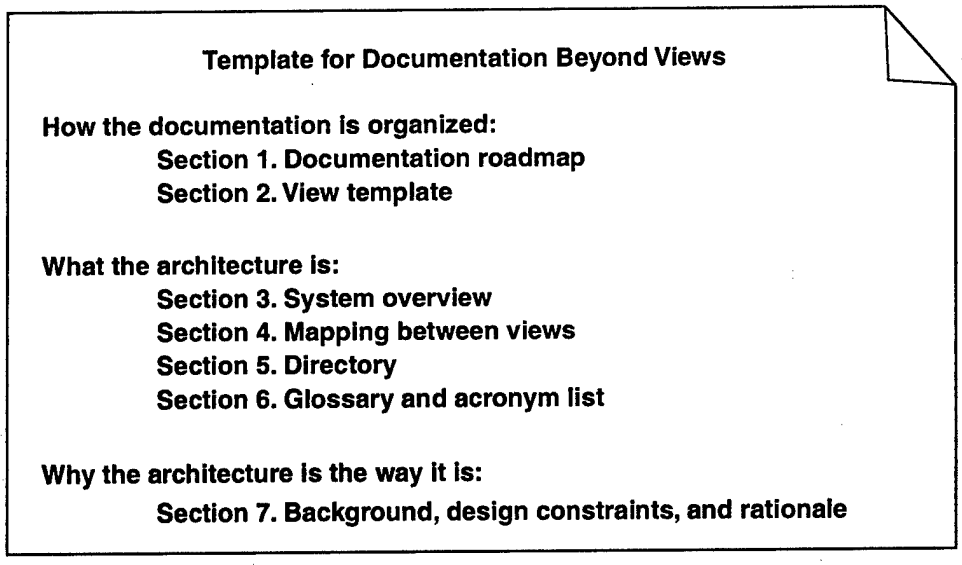


Figure 2: V&B Outline for Documentation Beyond Views

3 Agile Software Development

"Agile" refers to a paradigm of software development that emphasizes rapid and flexible development and de-emphasizes project and process infrastructure for their own sake. Figure 3 shows the so-called "manifesto" for agile software development, as articulated by the Agile Alliance, a non-profit organization "dedicated to promoting the concepts of agile software development and helping organizations adopt those concepts" [Agile Alliance 02b]. The manifesto includes among its signatories such luminaries as Kent Beck, Alistair Cockburn, Martin Fowler, Steve Mellor, and others.

A full treatment of agile methods is beyond the scope of this report, but books by Cockburn [Cockburn 02] and Beck [Beck 00] are foundation works.

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - **Individuals and interactions** over processes and tools
> - **Working software** over comprehensive documentation
> - **Customer collaboration** over contract negotiation
> - **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

Figure 3: The Manifesto for Agile Software Development [Agile Alliance 02a]

4 Examining the V&B and Agile Approaches

Clearly the V&B and agile paradigms start out from different philosophical vantage points. Can they be reconciled? If you want to use an agile approach for developing a system, is the V&B approach (or for that matter, *any* approach) for documenting software architectures to be summarily dismissed?

To answer this question, let us begin by focusing on the key underpinnings of each approach. On the agile side, the passage of interest with respect to the subject of this report is the one that says, "A working system is valued over comprehensive documentation." In addition, one of the stated principles of agile development is that "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation" [Agile Alliance 02c]. On the V&B side, it is useful to examine the three fundamental purposes behind architecture documentation:

1. Architecture serves as a means of education to introduce people to the system. Those people might be new members of the team, external analysts, or new architects.
2. Architecture serves as a primary vehicle for communication among stakeholders. An architecture's precise use as a communication vehicle depends on which stakeholders are doing the communicating. For instance, maintainers will use the documentation to measure the impact of a change and to identify the areas in which to begin work. Testers and integrators will use it to understand the desired black-box behavior of the system's elements and how they should fit together.
3. Architecture serves as the basis for system analysis. To support that analysis, the architecture documentation must contain the information necessary for the particular type of analysis being performed.

Agile developers discount these uses. Educating new people, they would say, is done by talking to the old people. Communication among stakeholders, they would say, is done through face-to-face conversation. Similarly, analysts can find out what they need to know simply by asking.

It doesn't take a vivid imagination to think of project scenarios in which relying on face-to-face communication is infeasible. Think of a performance analyst asking someone in the hallway what all of a system's 350 process deadlines are, so that the analyst can begin to compute

schedulability. Or think of a maintainer, who has inherited the system years after all the original developers have left, trying to understand where to begin. On a project involving hundreds of developers, do you really want your architects to spend all of their time answering the same questions over and over? Or would you rather let documentation serve that purpose, while also making sure the developers get the *same* answer every time?

In fact, agile development methods were invented for a particular context, and trying to apply them outside of their realm is unwise. In particular, there is no evidence that agile methods work for large, separately developed, long-lived systems that are turned over to different organizations for maintenance. In fact, the term *maintenance* is mentioned infrequently in the agile literature.[3]

One of the leading forces behind the agile movement, Alistair Cockburn, has done a superb job of explaining where the philosophy is and is not viable [Cockburn 02]. Life-critical systems, he says, are not feasible candidates for agile methods, nor are projects in which the team members are not collocated. His own agile method, Crystal, has an experience base that stops at about 40 participants [Cockburn 01]. Cockburn shows a graph whose x-axis is project size and whose y-axis denotes a system's criticality. He says that agile methods are clearly more at home in the lower left corner (small size projects on non-critical systems) of this space, and much less so elsewhere (larger size projects on more critical systems).[4]

It is no coincidence that these more challenging regions are exactly where software architecture itself plays its most critical role. One reason agile methods downplay architecture documentation is that they downplay architecture in favor of (at best) the detailed design of small pieces or (more often) code. On the other hand, while the V&B approach is ostensibly about architecture documentation, it brings with it the obligation to treat software architecture as a central concept for the system under development.

It is tempting to use Cockburn's table to divide systems into two classes: agile-prone and architecture-prone. But this is much too simplistic. On the one hand, many agile principles have clear value and appeal on any project:

- giving highest priority to satisfying the customer through early and continuous delivery of valuable software
- welcoming changing requirements, even late in development

---

3. Cockburn does, however, give future stakeholders their due. He writes, "Excessive documentation done too early delays the delivery of the software. If, however, too little documentation is done too late, the person who knows something needed for the next project has already vanished" [Cockburn 02].

4. Martin Fowler once asked if you would buy a Mazda Miata for its cargo carrying capability, implying that you would not use agile methods for large projects.

- delivering working software frequently, from every couple of weeks to every couple of months, with a preference to the shorter timescale
- business people and developers working together daily throughout the project
- building projects around motivated individuals
- Working software is the primary measure of progress.
- continuous attention to technical excellence and good design
- at regular intervals, reflecting on how to become more effective, then tuning and adjusting accordingly
- simplicity—the art of maximizing the amount of work *not* done

On the other hand, even small collocated non-life-critical systems can benefit from a disciplined approach to software architecture. Such an approach is the primary carrier of a system's quality attributes, the basis for analysis, and the medium for communication to the system's post-deployment stakeholders. The agile methods' oral tradition is insufficient for all these purposes.

And so we seek a middle ground. If you are beginning a project that lives in Cockburn's agile "sweet spot," you still might want to document your architecture to ensure the achievement of its critical quality attributes, to enable analysis, and to speak to future generations. However, if your project lives where the V&B approach holds sway, you still might want to try to bring some agile philosophy to bear. In either case, the questions for you become

- What architecture documentation do I need to produce?
- Can the V&B approach help, and if so, how?

The next section addresses these questions.

5 Reconciling the V&B and Agile Approaches

The V&B and agile philosophies agree strongly on a central point: If information isn't needed, don't document it. All documentation should have an intended use and audience in mind, and be produced in a way that serves both. One of the fundamental principles of technical documentation is "Write for the reader" [Clements 02]. That means understanding who will read the documentation and how they will use it. If there is no audience, there is no need to produce the documentation.

Architectural view selection is an example of applying this principle. The V&B approach, in concert with IEEE Std 1471-2000, prescribes producing a view if and only if it addresses the concerns of an explicitly identified stakeholder community.

Another central idea to remember is that documentation is not a monolithic activity that holds up all other progress until it is complete. Clements and associates prescribe producing the documentation in prioritized stages to satisfy the needs of the stakeholders who need it now [Clements 02, Ch. 9]. Cockburn expresses a similar idea this way: "The correct amount of documentation is exactly that needed for the receiver to make her next move in the game. Any effort to make the models complete, correct, and current past that point is a waste of money" [Cockburn 02]. The trick is knowing who the receivers are and what moves they need to make.

With that in mind, the following is the suggested approach for producing architecture documentation using agile-like principles:

1. Begin by creating a skeleton document for a comprehensive view-based software architecture document using the standard organization schemes shown in Figures 1 and 2. However, start with the outline only and leave the sections filled in initially with "to be determined."
2. Using the view selection scheme of the V&B approach, decide which architectural views you *would* want to produce, given enough resources [Clements 02, Ch. 9][5]. Choosing a view at this point does not obligate you to document it, but rather serves as a confirmation that there is a stakeholder community who will find information in that view useful, no matter how it is communicated. Choosing a view identifies a family of design decisions that the architect needs to resolve and be able to express. Add outlines for the chosen views to the outline you created in Step 1.

3. Annotate each section of the outline with a list of the stakeholders who should find the information it contains of benefit. Don't forget stakeholders who might not have joined the project yet, especially new hires, the maintenance staff, and successors to the current architect(s).

4. For sections that have an important stakeholder constituency *and* that you can fill in quickly using material at hand, do so. For example, a system overview available from other sources can be put to use easily. Or, the whiteboard sketches that agile methods prefer can be captured and put into the appropriate place(s) in the documentation skeleton.

5. Prioritize the completion of the remaining sections:

- If a section's constituency includes stakeholders for whom face-to-face conversation is impractical or impossible (e.g., maintainers in an as-yet-unidentified organization), that section will need to be filled in. If it includes *only* such stakeholders, its completion can be deferred until the conclusion of the project's development phase.[6]
- If a section's constituency includes *only* stakeholders for whom face-to-face conversation is practical and preferred, it may not need to be filled in. However, the architect may prefer filling it in to repeatedly answering the same questions about it. If a question about information in a particular section is asked, you can capture the question and answer it in that section. Thus, optional sections can become a list of frequently asked questions (FAQs) about the architecture that can be captured at a minimal cost.
- If a section's constituency includes both close-in and far-off constituents, try a combination of the approaches. Capture an FAQ list and convert it to a form more appropriate for archival purposes as time and resources permit.

We conclude this section with four of Cockburn's recommendations for documentation [Cockburn 02, pg. 177]:

- Bear in mind that there will be other people coming after this design team, people who will, indeed, need more design documentation.
- Run that as a parallel and resource-competing thread of the project instead of forcing it into the linear path of the project's development process.

---

5. This process consists of three steps. First, construct a table listing the stakeholders for the documentation as the rows, and the views that apply to the system as a column. Cells in the table designate whether a stakeholder needs to see a view in great detail, in some detail, in overview only, or not at all. This produces a set of candidate views. Second, combine those views in the candidate set that go well together. Third, prioritize and stage the remaining set as needed. The point of the selection process is to reduce the number of candidate views—which is almost always too large to be produced, kept consistent, and updated economically—to a manageable set. Getting the selected set of views requires the architect to not just guess at the stakeholders' needs, but rather to interact with stakeholders to make sure that their interests are being represented.

6. A caveat has to do with design rationale. Design rationale is almost always aimed at subsequent architects or maintainers to arm them with enough information to maintain the conceptual integrity of the system's design. Rationale, however, is *not* a dish best served cold. Waiting to capture rationale until the project winds down will certainly lead to the omission of key insights and thought patterns. Our advice is to capture rationale early and often.

- Be as inventive as possible about ways to reach the two goals adequately, dodging the impracticalities of being perfect.
- Find...the methodology...just rigorous enough that the communication is actually sufficient.

6 Conclusions

Agile development methods emphasize face-to-face communication over documentation. However, the very projects for which architecture serves the most important function—large, distributed, and long lived—are the projects for which face-to-face communication is the least practical. Even those projects that are a good fit for an agile approach—small, concentrated, and short lived—will benefit from carefully documenting the software architecture. In any case, agile methods bring certain positive aspects to a project, principally including the conscious decision about whether to produce artifacts.

This report has proposed a method for capturing architectural information in a manner consistent with agile philosophies. The method has, to our knowledge, not yet been applied in practice.

Both the V&B and agile approaches agree on one thing: Know why (and for whom) you are producing documentation before you set out to do it. The result should be an artifact that serves the needs of its constituency well and avoids superfluous effort and rework, thus making the effort to produce it worthwhile.

References

all URLS valid as of the publication date of this document

[Agile Alliance 02a] Agile Alliance. *Manifesto for Agile Software Development.* <http://www.agilemanifesto.org/> (2002).

[Agile Alliance 02b] Agile Alliance. <http://www.agilealliance.org> (2002).

[Agile Alliance 02c] Agile Alliance. *Principles Behind the Agile Manifesto.* <http://agilemanifesto.org/principles.html> (2002).

[Bachmann 02a] Bachmann, Felix; Bass, Len; Clements, Paul; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; & Stafford, Judith. *Documenting Software Architecture: Documenting Behavior* (CMU/SEI-2002-TN-001, ADA339792). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tn001.html>.

[Bachmann 02b] Bachmann, Felix; Bass, Len; Clements, Paul; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; & Stafford, Judith. *Documenting Software Architecture: Documenting Interfaces* (CMU/SEI-2002-TN-015, ADA403788). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tn015.html>.

[Bachmann 01] Bachmann, Felix; Bass, Len; Clements, Paul; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; & Stafford, Judith. *Documenting Software Architectures: Organization of Documentation Package* (CMU/SEI-2001-TN-010, ADA396052). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn010.html>.

[Bachmann 00] Bachmann, Felix; Bass, Len; Carrière, Jeromy; Clements, Paul; Garlan, David; Ivers, James; Nord, Robert; & Little, Reed. *Software Architecture Documentation in Practice: Documenting Architectural Layers* (CMU/SEI-2000-SR-004, ADA377988). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications /documents/00.reports/00sr004.html>.

[Beck 00] Beck, Kent. *Extreme Programming Explained: Embrace Change.* Reading, MA: Addison-Wesley, 2000.

[Clements 02] Clements, Paul; Bachmann, Felix; Bass, Len; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; & Stafford, Judith. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison-Wesley, 2002.

[Cockburn 02] Cockburn, Alistair. *Agile Software Development.* Boston, MA: Addison-Wesley, 2002.

[Cockburn 01] Cockburn, Alistair. *Crystal Methodologies.* <http://alistair.cockburn.us/crystal> (2001).

[IEEE 00] Institute of Electrical and Electronics Engineers. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (IEEE Std 1471-2000). Piscataway, NJ: IEEE Computer Press, 2000.

[Hofmeister 00] Hofmeister, Christine; Nord, Robert; & Soni, Dilip. *Applied Software Architecture.* Reading, MA: Addison-Wesley, 2000.

[Kruchten 00] Kruchten, Philippe. *The Rational Unified Process: An Introduction,* 2nd ed. Reading, MA: Addison-Wesley, 2000.

[Parnas 01] Parnas, D. L. "On a 'Buzzword:' Hierarchical Structure," Ch. 8, 161-170. *Software Fundamentals: Collected Papers by David L. Parnas.* Hoffman, D. M. & Weiss, D. M., eds. Boston, MA: Addison-Wesley, 2001.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (leave blank)	2. REPORT DATE July 2003	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Documenting Software Architectures in an Agile World		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Paul Clements, James Ivers, Reed Little, Robert Nord, Judith Stafford		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2003-TN-023
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12.a DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12.b DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

This report compares the Software Engineering Institute's Views and Beyond approach for documenting software architectures with the documentation philosophy embodied in agile software-development methods. This report proposes an approach for capturing architecture information in a way that is consistent with agile methods.

14. SUBJECT TERMS software architecture, architecture documentation, architecture views, agile methods			15. NUMBER OF PAGES 24 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102